CS305 – Programming Languages
Fall 2024-2025

HOMEWORK 2

Implementing a Syntax Analyzer (Parser) for Logical Circuit Designer (LCD)

Due date: October 30th, 2024 @ 23:55

NOTE

Only SUCourse submission is allowed. No submission by e-mail. Please see the note at the end of this document for late submission policy.

1 Introduction

In this homework, you will write a context-free grammar and implement a simple parser for the Logical Circuit Designer (LCD) language which you designed as a scanner in the previous homework. The language that will be generated by your grammar and other homework requirements are explained below.

2 Logical Circuit Designer (LCD) Language

The grammar you will design needs to generate the Logical Circuit Designer (LCD) language that is described below. Here is an example program written in this language. This is to give you an idea of what an LCD program looks like.

```
input X
node A, B
node C
output W, U
input Y, Z
output T

A = X or Y
B = A xor Y
C = Z and Y

W = A and (not Z)
U = B and C
```

```
T = A or not (C or B)

evaluate circuit1 (X = true, Y = true, Z = false)
evaluate circuit2 (X = false, Z = true, Y = false)
```

Here is the description of the syntactic rules of LCD language:

1. An LCD program consists of three consecutive blocks. These three blocks are the declarations block, the circuit design block, and the evaluations block. This order of blocks cannot be changed for a valid LCD program. That is we will always have first the declarations block, followed by the circuit design block, and finally the evaluations block. *An empty program is also considered an LCD program.*

2. The declarations block consists of any number of declarations. There are 3 different types of declarations: input declarations, output declarations, and node declarations. An input declaration starts with the keyword `input`, a node declaration starts with the keyword `node`, and an output declaration starts with the keyword `output`. After the keyword of the declaration, an identifier list is given. An identifier list is either a single identifier or a comma-separated list of identifiers.

   Note that different type of declarations can be interleaved. That is, input, node, and output declarations can be given in any order and there can me more than one input/node/output declarations (as demonstrated in the example above).

   An example declaration block can be seen in the first 6 lines of the example given above.

3. The circuit design block consists of a list of assignments. which can be empty. Each assignment is formed in the following way; an identifier is given, this is followed by the assignment operator, and finally an expression is given (e.g. `A = X or Y`).

4. An expression is either a single identifier, `not` operator applied to an expression, two expressions combined with a binary logical operator (which can be one of `and`, `or`, `xor`) between them, or a constant boolean value (`true` or `false`). An expression can be surrounded by a pair of parenthesis.

5. In terms of operator precedence and associativity, not operator has the highest priority. The second highest priority belongs to and operator. At the lowest priority level, we have `or` and `xor` operators. All binary operators are left–associative.

6. The evaluations block consists of evaluations. An evaluation is formed in the following way; first, the keyword `evaluate` is given, followed by an identifier, and an input–initialization list is given between a pair of parenthesis (please see the last 2 lines of the example above, where there are 2 evaluate statements in the evaluations block). An input–evaluation list is either a single input–evaluation or comma-separated list of input–evaluations. An input–evaluation is an identifier followed by an assignment symbol, which is followed by a constant boolean value (`true` or `false`).

3 Terminal Symbols

Although you can implement your own scanner, we provide a flex scanner for this homework. The provided flex scanner implements the following tokens.

`tINPUT`: The scanner returns this token when it sees `input`

`tOUTPUT`: The scanner returns this token when it sees `output`

`tNODE`: The scanner returns this token when it sees `node`

`tEVALUATE`: The scanner returns this token when it sees `evaluate`

`tIDENTIFIER`: The scanner returns this token when it sees an identifier

`tAND`: The scanner returns this token when it sees `and`

`tOR`: The scanner returns this token when it sees `or`

`tXOR`: The scanner returns this token when it sees `xor` in the input.

`tNOT`: The scanner returns this token when it sees an `not` in the input.

`tTRUE`: The scanner returns this token when it sees a `true`

`tFALSE`: The scanner returns this token when it sees a `false`

`tLPR`: The scanner returns this token when it sees `(`

`tRPR`: The scanner returns this token when it sees `)`

`tASSIGNMENT`: The scanner returns this token when it sees `=`

`tCOMMA`: The scanner returns this token when it sees a comma

Besides these tokens, the scanner silently consumes white space characters. Any other character which is not recognized as part of a lexeme of a token is returned to the parser.

These tokens and their lexemes are explained in the Homework 1 document.

4 Output

Your parser must print out `OK` and produce a new line if the input is grammatically correct. Otherwise, your parser must print out `ERROR` and produce a new line. In other words, the main part in your parser file must be as follows (and there should be no other part in your parser that produces an output):

```
int main ()
{
   if (yyparse())
   {
      // parse error
      printf("ERROR\n");
      return 1;
   }
   else
   {
      // successful parsing
      printf("OK\n");
      return 0;
   }
}
```

In short, if the file `test1.lcd` includes a grammatically correct LCD program then your output should be `OK`, and otherwise, your output should be `ERROR`.

5 How to Submit

Submit your Bison file named as `username-hw2.y`, and flex file named as `username-hw2.flx` (note that, even if you use the flex file we provide, you still need to submit it after

renaming it as indicated here), where `username` is your SU-Net username and **do not zip your files.** We will compile your files by using the following commands:

```
flex username-hw2.flx
bison -d username-hw2.y
gcc -o username-hw2 lex.yy.c username-hw2.tab.c -lfl
```

So, make sure that these three commands are enough to produce the executable parser. If we assume that there is a text file named test1.lcd, we will try out your parser by using the following command line:

```
username-hw2 < test1.lcd
```

6 Notes

- **Important**: Name your files as you are told and **don't zip them.** [-10 points otherwise]
- **Important**: Make sure you include the correct "`...tab.h`" file in your scanner and make sure you can compile your parser using the commands given in Section 5. If we are not able to compile your code with those commands **your grade will be zero for this homework.**
- **Important**: Since this homework is evaluated automatically make sure your output is exactly as it is supposed to be. (i.e. OK for grammatically correct LCD programs and ERROR otherwise).
- No homework will be accepted if it is not submitted using SUCourse.
- You may get help from our TA or from your friends. However, **you must write your bison file by yourself**.
- Start working on the homework immediately.
- If you develop your code or create your test files on your own computer (not on cs305.sabanciuniv.edu), there can be incompatibilities once you transfer them to the cs305 machine. Since the grading will be done automatically on the cs305 machine, we strongly encourage you to do your development on the cs305 machine, or at least test your code on the cs305 machine before submitting it. If you prefer not to test your implementation on the cs305 machine, this means you accept to take the risks of incompatibilities. Even if you may have spent hours on the homework, you can easily get 0 due to such incompatibilities.

LATE SUBMISSION POLICY

Late submission is allowed subject to the following conditions:

- Your homework grade will be decided by multiplying what you get from the test cases by a "submission time factor (STF)".
- If you submit on time (i.e. before the deadline), your STF is 1. So, you don't lose anything.
- If you submit late, you will lose 0.01 of your STF for every 5 minutes of delay.
- We will not accept any homework later than 500 minutes after the deadline.
- SUCourse's timestamp will be used for STF computation.
- If you submit multiple times, the last submitted version and its submission time will be used.